Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

Collares perfectos máximos

Tesis de Licenciatura en Ciencias de la Computación

Ezequiel Zimenspitz

Director: Verónica Becher

Buenos Aires, 16 de Julio de 2020

COLLARES PERFECTOS MÁXIMOS

Hay secuencias muy lindas. Otras aún más lindas. Pero hay algunas que son combinatoriamente perfectas. Estas se vuelven atractivas en la práctica si además podemos dar un algoritmo sencillo y rápido para generarlas. En esta tesis nos dedicamos a unas de estas secuencias combinatoriamente perfectas, variantes de las bien conocidas secuencias de Bruijn. Mostramos que el algoritmo goloso de Fredricksen de 1982 para generar las secuencias de Bruijn lexicográficamente máximas se puede adaptar para generar estas secuencias perfectas lexicográficamente máximas.

MAXIMAL PERFECT NECKLACES

There are very nice sequences. Others even nicer. But there are some that are combinatorially perfect. These become attractive in practice if we can give a simple and fast algorithm to generate them. In this thesis we focus on one of these combinatorially perfect sequences, variants of the well-known de Bruijn sequences We show that Fredricksen's greedy algorithm of 1982 to generate the lexicographically maximal de Bruijn sequences can be adapted to generate these lexicographically maximal perfect sequences.

A Vero, que siempre estuvo atenta y dispuesta a ayudarme durante todo este proceso.

A Agus, Chamo, Dona, Fabro y Fran, con quienes tuve la suerte de compartir muchos tps, horas de estudio y por sobre todo, muchas risas.

A mi equipo de Básquet de Exactas, con quienes espero seguir compartiendo cancha muchos años más.

A mi familia, que me banca en todas, siempre.

CONTENTS

1. INTRODUCTION

Fix a finite alphabet $\mathcal{A}$ and write $|\mathcal{A}|$ for its cardinality. A word is a finite sequence of symbols in the alphabet. A rotation is the operation that moves the final symbol of a word to the first position while shifting all other symbols to the next position, or it is the composition of this operation with itself an arbitrary number of times. A circular word, or necklace is the equivalence class of a word under rotations.

Definition (de Bruijn necklace)**.** A de Bruijn necklace [3] of order k in $\mathcal{A}$ is a necklace of length $|\mathcal{A}|^k$ in which each word of length k occurs exactly once as a contiguous subsequence.

Example. For $\mathcal{A} = \{0, 1\}$ and $k = 2$, [0011] is a de Bruijn necklace, but [0101] is not.

A total order on the symbols in the alphabet induces a total order of words of the same length, with in turn induces a total order on necklaces as follows: a necklace is greater than other if there is an element in the equivalent class of the first one that is greater than all the elements in the equivalent class of the second one. For example [11100100] is greater than [11011000]. Thus, for each order k there is a maximal de Bruijn necklace. The lexicographically least de Bruijn sequence of a given order is known as the Ford sequence [4]. In the sequel we consider the necklace defined by the Ford sequence, and refer to it as the Ford necklace.

Example (Minimal and Maximal de Bruijn necklaces)**.**

For $k = 1$ and $k = 2$, the minimal and maximal necklaces are the same because they belong to the same equivalent class.

$k = 1 \rightarrow [01]$

$k = 2 \rightarrow [0011]$

$k = 3 \rightarrow$ minimal: [00010111], maximal: [00011101]

Fix the alphabet $\{0, 1\}$, fix a positive integer k and consider all the words of length k, in lexicographic order. For example,

$$000 \;\; 001 \;\; 010 \;\; 011 \;\; 100 \;\; 101 \;\; 110 \;\; 111$$

Now consider the concatenation of these 8 words and consider the resulting necklace

$$[000001010011100101110111]$$

This necklace has the striking property that all words of length 3 occur exactly 3 times, but each time occur at a different position modulo 3, for any convention on the starting position. Motivated by this example Alvarez, Becher, Ferrari and Yuhjtman in [7] formalized the notion of perfect necklaces, which are variants of the de Bruijn necklaces.

Definition ((k, n)-perfect necklace)**.** A necklace is (k, n)-perfect if it has length $n|\mathcal{A}|^k$ and each word of length k occurs exactly n times at positions which are different modulo n for any convention on the starting point.

Remark. For any alphabet size, a $(k, 1)$-perfect necklace is a de Bruijn necklace.

Definition (Perfect necklace)**.** A necklace is perfect if it is (k, k)-perfect for some k.

Example (Perfect Necklace)**.** Let $\mathcal{A} = \{0, 1\}$. We add spaces in the examples just for readability. For words of length 2 there are just two perfect necklaces:
[00 01 10 11],
[00 10 01 11].

This is a perfect necklace for word length 3:
[000 110 101 111 001 010 011 100].

The following are not perfect:
[00 01 11 10] since 00 appears in position 1 and 7 which are equal modulo 2, but it also appears a third time in position 2,
[000 001 011 010 110 111 101 100] since 101 appears in position 5 and 8 which are equal modulo 3.

Each de Bruijn necklace of order k in $\mathcal{A}$ is the label of a Hamiltonian cycle in a de Bruijn graph of order k. Since the line graph of the de Bruijn graph of order k is the de Bruijn graph of order $k-1$, each de Bruijn necklace of order k is the label of an Eulerian cycle in the de Bruijn graph of order $k-1$. The number of distinct de Bruijn sequences of order k in an alphabet of b symbols is given by the formula [3]

$$\frac{(b!)^{b^{k-1}}}{b^k}$$

Thanks to these properties of the de Bruijn graphs, it is possible to extend a de Bruijn necklace of order k to a de Bruijn necklace of order $k+1$, see [1].

It is also possible to count the number of perfect necklaces of order k in an alphabet of b symbols [7].

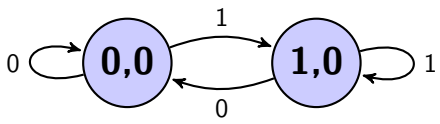
Becher and Carton studied a special class of perfect necklaces called nested perfect necklaces [2]

Definition (Astute graph)**.** Let $\mathcal{A}$ be an alphabet with cardinality b, let k be a word length and let n be a positive integer. The astute graph $G_{k,n}$ is a directed graph, with nb^k nodes, each node is a pair (u, v), where u is in $\mathcal{A}^k$ and v is a number between 0 and $n-1$. There is an edge from (u, v) to (u', v') if the last $k-1$ symbols from u coincide with the first $k-1$ symbols from u' and $(v+1) \bmod n = v'$.

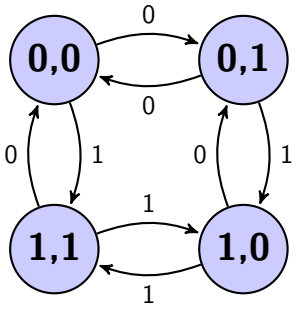
Perfect necklaces of order k are the label of Hamiltonian cycles in the astute graphs of order k. Equivalently, they are the label of Eulerian cycles in astute graphs of order $k-1$.

Example (Astute graph)**.** Let $\mathcal{A} = \{0, 1\}$.

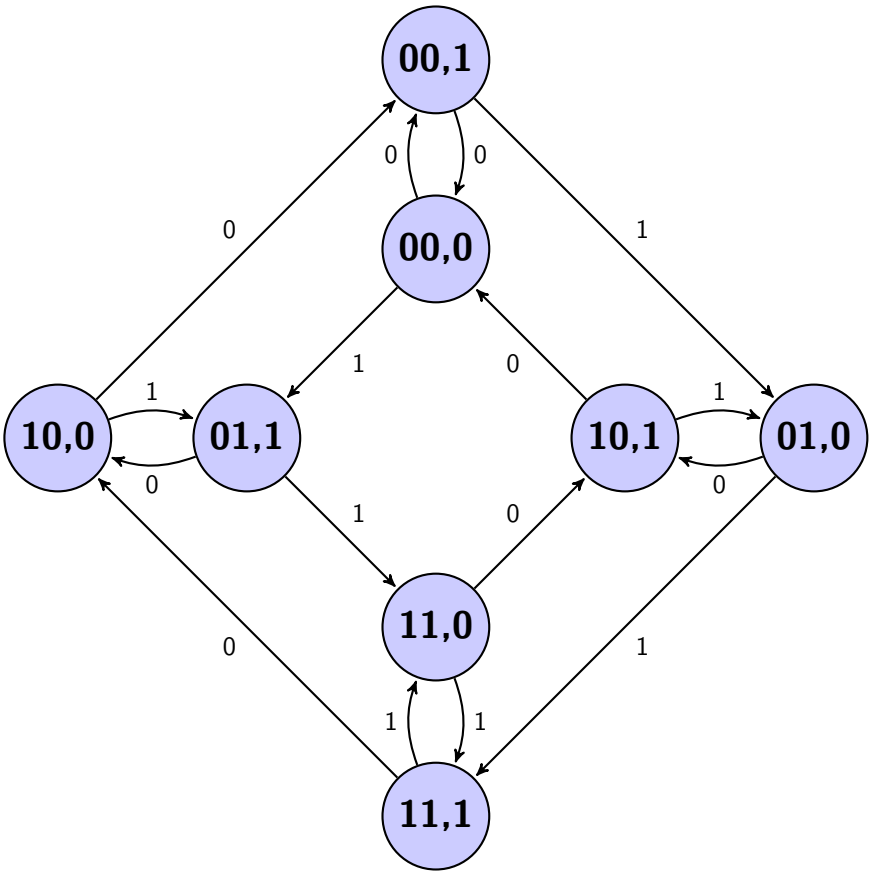
$G_{1,1} =$



$G_{1,2} =$



$G_{2,2} =$



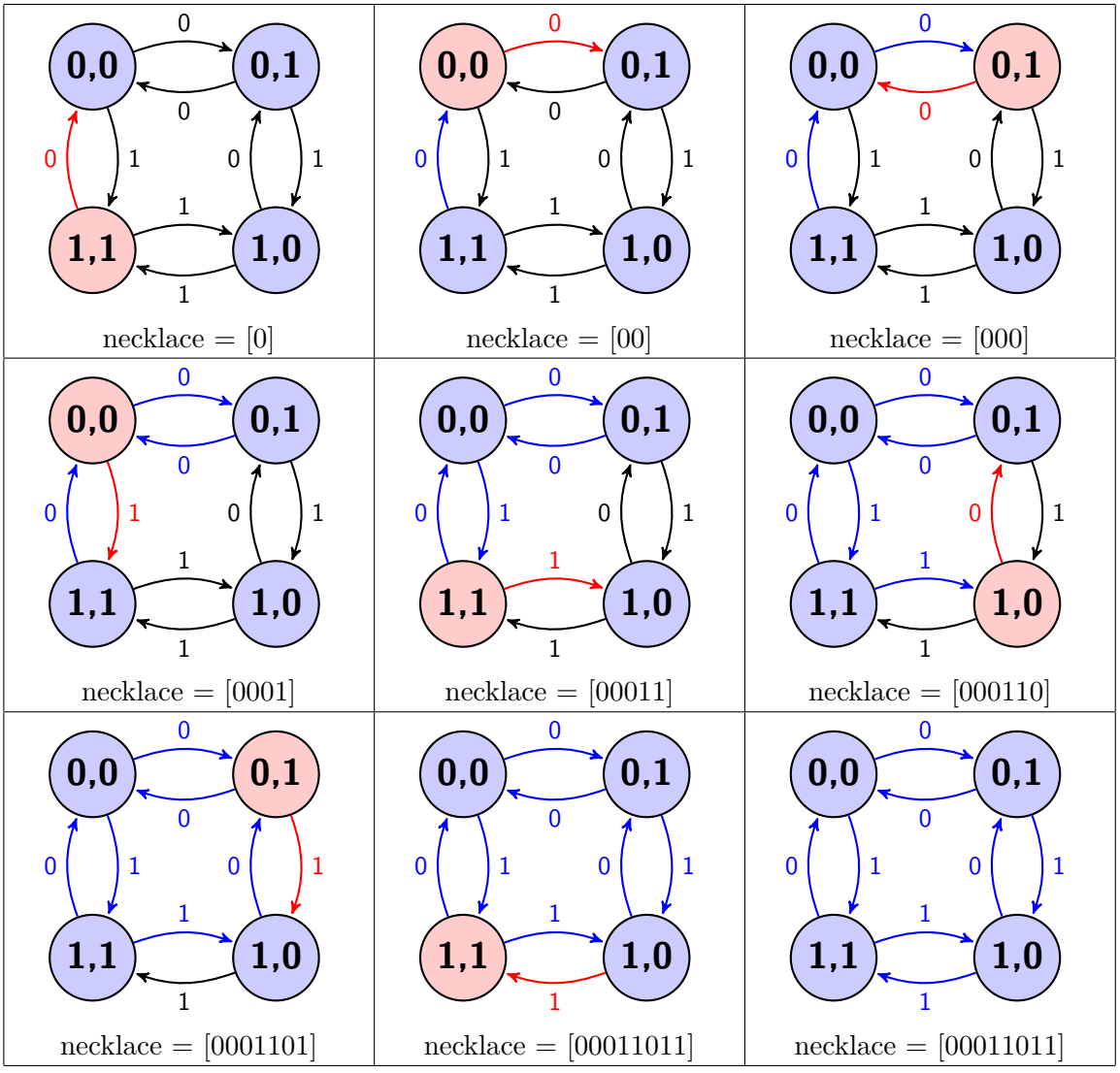
Notice that the astute graphs for $n > 1$ have no self loops.

Remark. For any alphabet size, the astute graph $G_{k-1,1}$ coincides with a de Bruijn graph

of words of length $k-1$; hence, the Eulerian cycles in $G_{k-1,1}$ yield exactly the de Bruijn necklaces of order k.

Remark. Each Eulerian cycle in the astute graph $G_{k-1,n}$ gives one (k,n)-perfect necklace [7].

Example (Eulerian cycle in $G_{1,2}$)**.** Starting at $(1,1)$; current node in red and moving in the red edge; used edges in blue:



2. ALGORITHMS

There are many of algorithms in the literature to generate de Bruijn necklaces, some of them can be found in the survey by Fredricksen in 1982 [5].

The simplest algorithm given by Fredricksen in [5] is the Prefer One algorithm and it generates the lexicographically greatest de Bruijn necklace.

2.1 Prefer One Algorithm for de Bruijn necklaces (Fredricksen 1982)

1. Start with a sequence of k zeros.
2. If the sequence has length 2^k stop; otherwise,
3. For the ith bit of the sequence, $i > k$, write a one if the newly formed k-tuple has not previously appeared in the sequence; otherwise write a zero. Increase i and repeat step 2;

Theorem (Fredericksen 1982 [5])**.** *Prefer One is a greedy algorithm that generates a de Bruijn necklace of order k with alphabet $\mathcal{A} = \{0, 1\}$ that has a running time linear in the size of the output.*

Proof. In 1736, Euler showed that a directed graph G has an Eulerian cycle if and only if G is connected, and the indegree is equal to the outdegree at every node. In this case G is called Eulerian. The BEST theorem for de Bruijn, Ehrenfest, Smith and Tutte [6], states that the number of Eulerian cycles in a connected Eulerian graph can be obtained by counting the number of rooted trees which are subgraphs of a given graph $G = (V, E)$. A tree is a connected graph without cycles. The number of edges in a tree is therefore one less than the number of vertices in the tree. A root of the tree is a node which is not the initial node of any edge (we think here of directed trees).

For a given graph $G = (V, E)$ the BEST theorem determines the number of subgraphs of G which are rooted trees. We define the associated matrix A_G of size $|V| \times |V|$ as

$A_{G_{ii}}$ is the number of edges exiting the node numbered i.
$A_{G_{ij}}$ is the negative of the number of edges from node i to node j.

Then the number of trees rooted at i is given by the determinant of the minor of the element $A_{G_{ii}}$.

As an example, consider the de Bruijn graph of order 3 $G_{3,1}$, in Figure 2.2. Since the theorem requires a graph without loops, we drop the loop at nodes 000 and 111. (Removing these loops does not alter the number of trees which are subgraphs of $G_{3,1}$.)

The associated matrix for a de Bruijn graph of order k, that we will call A_{G_k}, has the form: Row 0 has a 1 in column 0 and -1 in column 1. Row i has a 2 in column i, a -1 in column $2i$ and a -1 in column $2i+1$ (modulo 2^k) for $i = 1, 2, ..., 2^k - 2$. Row $2^k - 1$ has a 1 in column $2^k - 1$ and a -1 in column $2^k - 2$.

Then we compute the determinant of the minor of the $(0, 0)$ position of A_{G_k} to get the number of rooted trees in G_k, with root 0^k. It can be shown by induction that there are $2^{2^k-(k+1)}$ trees with root 0^k.

From the BEST theorem it can be derived the following algorithm that can traverse a de Bruijn graph forming an Eulerian cycle.

1. Find a tree rooted at 0^{k-1} in G_{k-1}
2. Add the loop at node 0^{k-1} to any tree rooted at 0^{k-1} in G_{k-1}, place a star on it.
3. Place a star on all edges of the tree.
4. Choose node 0^{k-1} as the starting node
5. When at node i, leave the node by the unstarred edge unless that edge has been used. Otherwise leave by the starred edge,
6. Determine the state of the process by the label of the edge traversed.
7. When we enter a node, if the starred edge out of that node has been used we stop. Otherwise return to step 5.

Fredricksen uses the tree rooted at 0^{k-1} formed by the edges with label 0. The sequence obtained in the Eulerian cycle is the same sequence we obtain with the Prefer One algorithm.

Analyzing the running time, in each iteration of the loop the algorithm does a constant number of operations, and each decision is local with respect to the node being visited.

In order to make a decision it needs to verify if the newly formed k-tuple was already written in the output. So it maintains a set in which, if the word exist, then it was written in the output, otherwise, it was not. At each loop iteration:

1. the algorithm does one lookup in the set and chooses 1 or 0.
2. writes the decision in the output.
3. writes the set with the newly formed k-tuple.

We know that there are 2^k possible words of length k with 2 letters, so there will be 2^k loop iterations at most. Therefore, the algorithm has a running time linear in the size of the output.

Example ($G_{3,1}$ rooted tree at 000)**.**

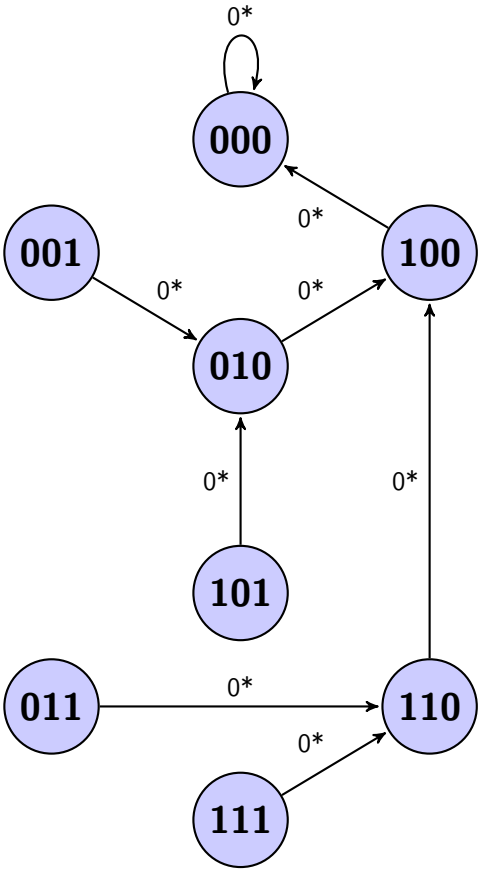


Fig. 2.1: Tree with root at 000 and the loop in 000

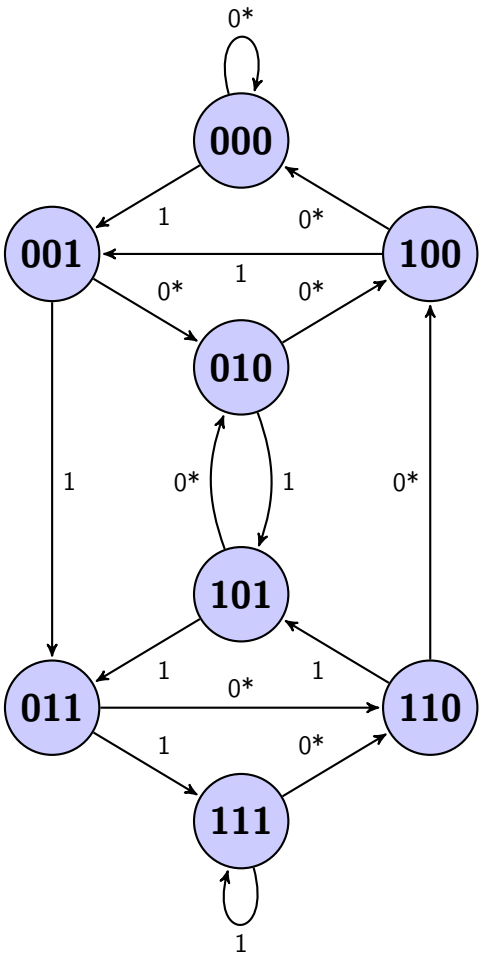


Fig. 2.2: $G_{3,1}$

Example (de Bruijn necklaces)**.**

$k = 1 \rightarrow [01]$

$k = 2 \rightarrow [0011]$

$k = 3 \rightarrow [00011101]$

2.2 Prefer One Algorithm for Perfect Necklaces

We adapt Fredricksen's Prefer One algorithm to produce the lexicographically greatest perfect necklace of order k with alphabet $\mathcal{A} = \{0, 1\}$.

We are going to verify that a newly formed k-tuple has not previously appeared in the

same position mod k.

1. Start with a sequence of k zeros.
2. If the sequence has length $k * 2^k$ stop; otherwise,
3. For the ith bit of the sequence, $i > k$, write a one if the newly formed k-tuple either has not occurred yet in the sequence, or it has not occurred at a position j such that j mod k is equal than i mod k. otherwise write a zero. Increase i and repeat step 2;

Theorem. *Prefer One is a greedy algorithm that generates a perfect necklace of order k with alphabet $\mathcal{A} = \{0, 1\}$ that has a running time linear in the size of the output.*

Proof. For $k = 1$, the proof remains the same as de Bruijn algorithm, since de Bruijn graph and astute graph are equivalent.

For $k > 1$, consider the following algorithm:

1. Find a tree rooted at $(0^{k-1}, 0)$ in $G_{k-1,k}$
2. Place a star on the edge with label 0 that goes from node $(0^{k-1}, 0)$ to node $(0^{k-1}, 1)$.
3. Place a star on all edges of the tree.
4. Choose node $(0^{k-1}, 0)$ as the starting node
5. When at node i, leave the node by the unstarred edge unless that edge has been used. Otherwise leave by the starred edge,
6. Determine the state of the process by the label of the edge traversed.
7. When we enter a node, if there are no starred edges out of that node that can be used we stop. Otherwise return to step 5.

To proof this algorithm works in general, we need to show that each edge is used once and that we terminate at node $(0^{k-1}, 0)$.

If the algorithm stops at any node it must have exited that node k times earlier, but when we do stop, we have just visited the node for the $(k+1)th$ time. This can only occur if the final node is the initial node $(0^{k-1}, 0)$.

If some edge is not used, we can assume it is starred. Since the original tree had no loops, there is a path starting from this starred edge and leading to the edge with label 0 that exits the node $(0^{k-1}, 0)$ along edges all of which are starred. (If one of these starred edges is unused, all of them must be.) Thus, the algorithm will not have terminated in node $(0^{k-1}, 0)$, but it does, so this contradiction proves the algorithm.

Therefore, using the tree rooted at $(0^{k-1}, 0)$ formed by the edges with label 0, the sequence obtained in the Eulerian cycle is the same sequence we obtain with the Prefer One algorithm.

Analyzing the running time, in each iteration of the loop the algorithm does a constant number of operations, and each decision is local with respect to the node being visited.

In order to make a decision it needs to verify if the newly formed k-tuple was already written in the output in a position mod k. So it maintains a multimap with k-tuples as keys and a 0 and/or a 1 as values, a 0 if the k-tuple was written in the output for position 0 mod k, a 1 if it was written for position 1 mod k. At each loop iteration:

1. the algorithm does one lookup in the set and chooses 1 or 0.
2. writes the decision in the output.
3. writes the map with the newly formed k-tuple and its position mod n.

We know that there are $k * 2^k$ possible words of length k with 2 letters in k different positions, so there will be $k * 2^k$ loop iterations at most. Therefore, the algorithm has a running time linear in the size of the output.

Example ($G_{2,2}$)**.**

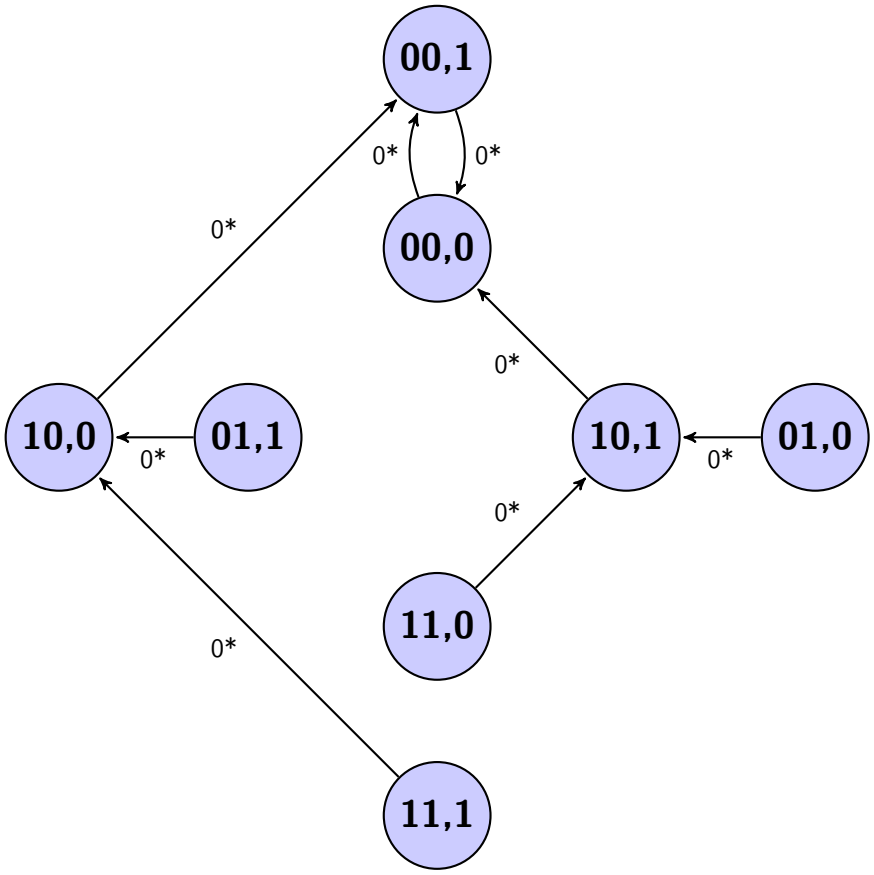


Fig. 2.3: Tree with root at $(00, 0)$ and the edge 0 from node $(00, 0)$

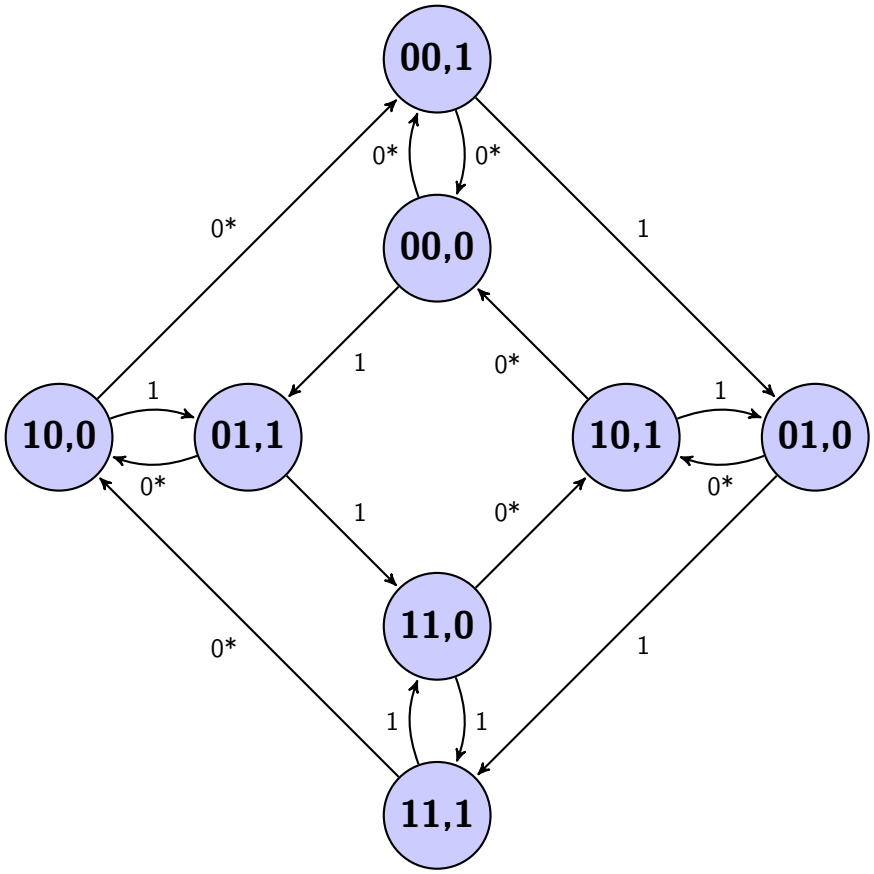


Fig. 2.4: $G_{2,2}$

Example (Perfect Necklaces)**.**

$k = 1 \rightarrow [01]$

$k = 2 \rightarrow [00111001]$

$k = 3 \rightarrow [000111110101100011010001]$

Theorem. *Prefer One generates perfect necklaces of order k that are lexicographically greatest.*

Proof. In every loop iteration we try to write the lexicographically greatest character in the alphabet, therefore, the necklace will be the greatest possible.

2.3 Prefer Max Algorithm

Now we are going to extend this algorithm to make perfect necklaces of order k with alphabet $\mathcal{A} = \{0, 1, ..., b-1\}$ for any integer value of b greater than 1.

1. Start with a sequence of k zeros.

2. If the sequence has length $k * b^k$ stop; otherwise,

3. For the ith bit of the sequence, $i > k$.

   Select the higher p from $b-1$ to 0 that will make the newly formed k-tuple either not occurred yet in the sequence, or not occurred at a position j such that j mod k is equal than i mod k.

   Write p. Increase i and repeat step 2;

Theorem. *Prefer Max is a greedy algorithm that generates a perfect necklace of order k with alphabet $\mathcal{A} = \{0, 1, ..., b-1\}$ that has a running time linear in the size of the output times b.*

Theorem. *Prefer Max generates perfect necklaces of order k that are lexicographically greatest.*

To proof the theorems, the idea is the same as for the Prefer One algorithm, but trying to use the greatest label in each step.

Remark. The Prefer Min version is identical but starting with a sequence of k $b-1$s and j iterating from 0 to $b-1$. This version will generate the lexicographically least necklace.

Remark. The algorithm can be slightly modified to generate (k, n)-perfect necklaces with the same strategy.

3. EXAMPLES

Here we show some outputs generated by the algorithms.

$\mathcal{A} = \{0, 1\}$

$k = 1 \rightarrow [01]$

$k = 2 \rightarrow [00111001]$

$k = 3 \rightarrow [000111110101100011010001]$

$k = 4 \rightarrow [0000111111101101110010111010100110000111011001010100001100100001]$

$\mathcal{A} = \{0, 1, 2\}$

$k = 1 \rightarrow [021]$

$k = 2 \rightarrow [002221201211100201]$

$k = 3 \rightarrow [000222221220212211210202201200122121120112111110102101100022021020012011010002001]$

$k = 4 \rightarrow [000022222221222022122211221022022012200212222121212021122111211021022101210020222021202020122011201020022001200012222122112201212121112101202120112001122112111201112111111101102110111001022102110201012101110101002100110000222022102200212021102100202020102000122012101200112011101100102010101000022002100200012001100100002000001]$

$\mathcal{A} = \{0, 1, 2, 3, 4\}$

$k = 1 \rightarrow [04321]$

$k = 2 \rightarrow [00444342414034333231302423222120141312111004030201]$

$k = 3 \rightarrow [000444443442441440434433432431430424423422421420414413412411410404403402401400344343342341340334333332331330324323322321320314313312311310304303302301300244243242241240234233232231230224223222221220214213212211210204203202201200144143142141140134133132131130124123122121120114113112111110104103102101100044043042041040034033032031030024023022021020014013012011010004003002001]$

$k = 4 \rightarrow [000044444443444244414440443444334432443144304424442344224421442044144413441244114410440440344024401440043444343434243414340433443334332433143300

43244323432243214320431443134312431143104304430343024301430042444243424242414240423443342342314230422442234222422142204214421342124211421042044203420242014200414441434142414141404134413341324131413041244123412241214120411441134112411411041044103410241014100404440434042404140404034403340324031403040244023402240214020401440134012401140104004400340024001400034443443344234413440343434333432431343034243423342234213420341434133412341134103404340334023401340033443343342334133403333433333332333133303324332333223321332033143313331233113310330433033302330133003244324332423241324032343233332323231323032243223322232213220321433213212321132103204320332023201320031443143314231413140313431333132313131303124312331223121312031143113311231113110310431033102310131003044304330423041303043033032303130303024302330223021302030143013301230113010300430033002300130002444244324422441244024342433243224312430242424232422242124202414241324122411241024042403240224012400234423432342234123402334233332332331233023242323232223212320231423132312231123102304230323022301230022442243224222412240223422332232223122302224222322222221222022142213221222112210220422032202220122002144214321422141214021342133213221312130212421232122212121202114211321122111211021042103210221012100204420432042204120402034203320322031203020242023202220212020201420132012201120102004200320022001200014441443144214411440143414331432143114301424142314221421142014141413141214111410140414031402140114001344134313421341134013341333133213311330132413231322132113201314131313121311131013041303130213011300124412431242124112401234123312321231123012241223122212211220121412131212121112101204120312021201120011441143114211411140113411331132113111301124112311221121112011141113111211111110110411031102110111001044104310421041104010341033103210311030102410231022102110201014101310121011101010041003100210011000044404430442044104400434043304320431043004240423042204210420041404130412041104100404040304020401040003440343034203410340033403330332033103300324032303220321032003140313031203110310030403030302030103000244024302420241024002340233023202310230022402230222022102200214021302120211021002040203020202010200014401430142014101400134013301320131013001240123012201210120011401130112011101100104010301020101010000440043004200410040003400330032003100300024002300220021002000140013001200110010000400030002000 1]

Now we show some perfect necklaces as images in gray scale.

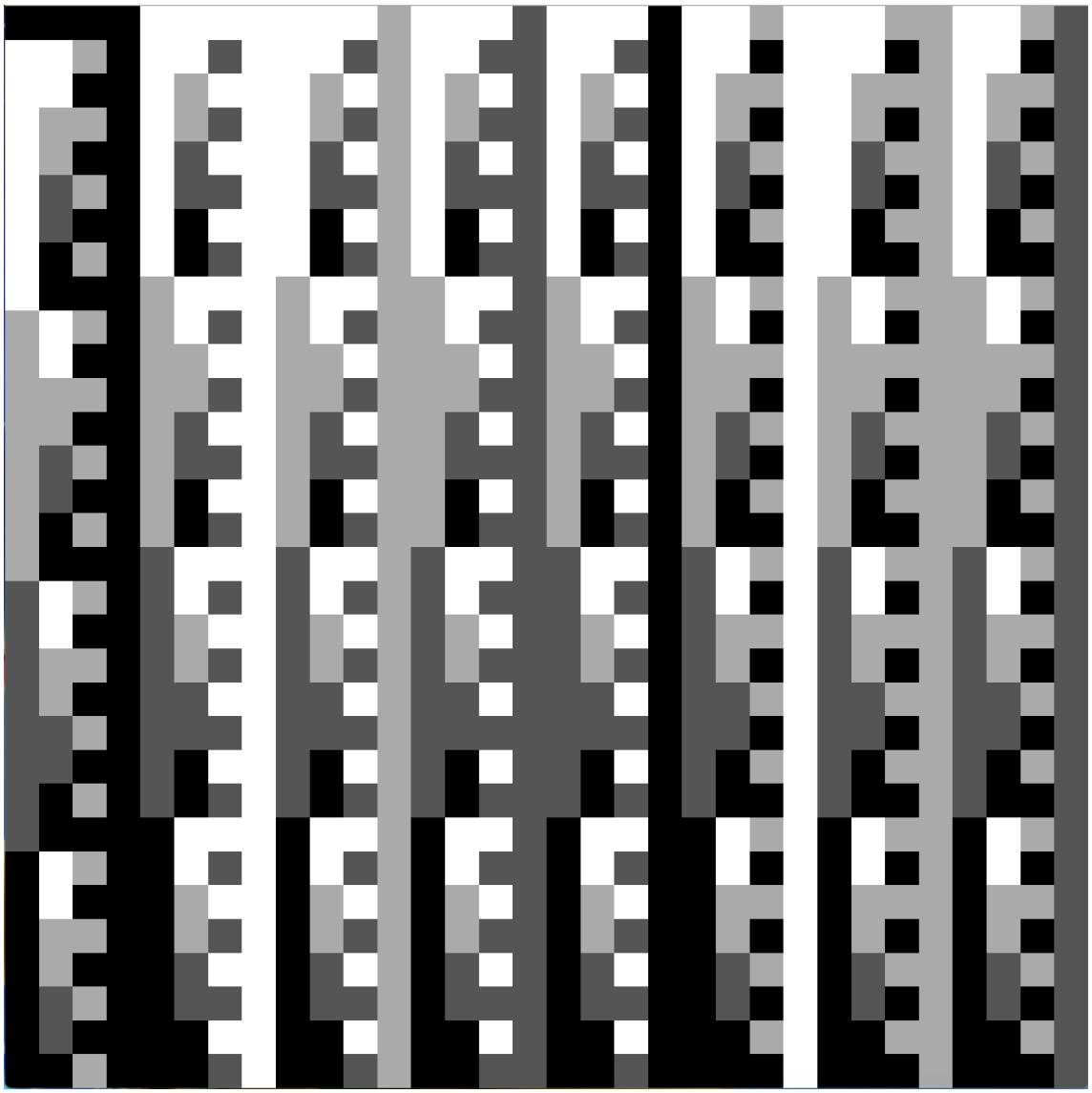


Fig. 3.1: Perfect necklace with $\mathcal{A} = \{0, 1, 2, 3\}$ and $k = 4$. $\mathcal{A}$ and k were selected only so that $k|\mathcal{A}|^k$ is a perfect square. This is a plot of the sequence in a matrix of 32×32. Position (i, j) in the matrix depicts the symbol at position $32 \times i + j$ in the sequence. Black for 0s, dark gray for 1s, light gray for 2s, white for 3s.

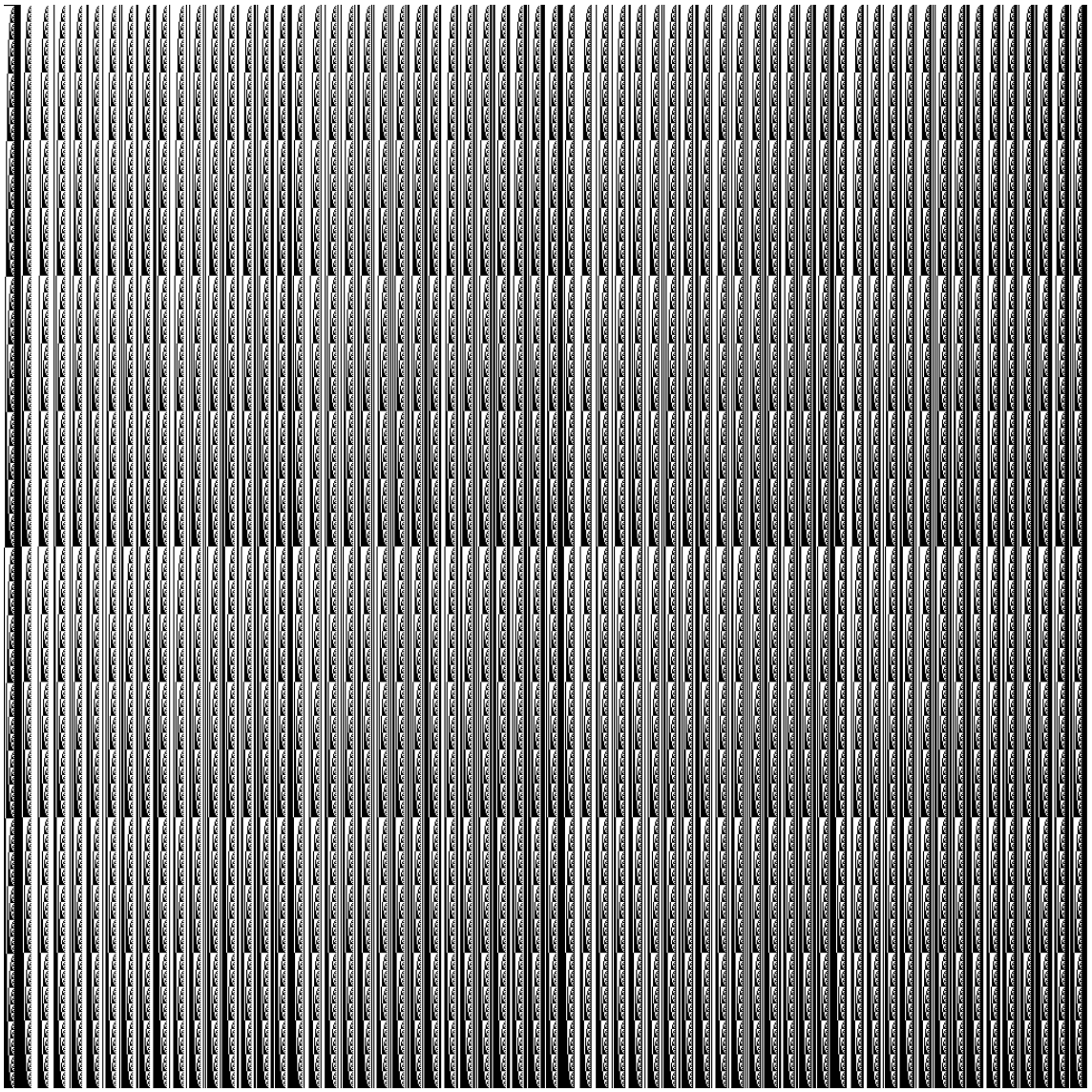


Fig. 3.2: Perfect necklace with $\mathcal{A} = \{0, 1\}$ and $k = 16$. $\mathcal{A}$ and k were selected only so that $k|\mathcal{A}|^k$ is a perfect square. This is a plot of the sequence in a matrix of 1024×1024. Position (i, j) in the matrix depicts the symbol at position $1024 \times i + j$ in the sequence. Black for 0s, white for 1s.

4. APPENDIX

Prefer One

```
def generate(self, k):
    expected_length = k * 2 ** k
    necklace = self.__zero * k

    apparitions_mod_k = {necklace: [0]}
    current_position = len(necklace) - k + 1
    for _ in range(expected_length - k):
        current_position_mod_k = current_position % k
        next_word = necklace[current_position:] + self.__one
        if not (next_word in apparitions_mod_k and current_position_mod_k in apparitions_mod_k[next_word]):
            necklace += self.__one
        else:
            necklace += self.__zero
        if next_word not in apparitions_mod_k:
            apparitions_mod_k[next_word] = []
        apparitions_mod_k[next_word].append(current_position_mod_k)
        current_position += 1
    return necklace
```

Prefer Max

```
def generate(self, k):
    letters = self.__alphabet.get_letters()
    expected_length = k * len(letters) ** k
    necklace = letters[0] * k

    apparitions_mod_k = {necklace: [0]}
    current_position = len(necklace) - k + 1
    for i in range(expected_length - k):
        current_position_mod_k = current_position % k
        next_word = ''
        for letter in reversed(letters):
            next_word = necklace[current_position:] + letter
            if not (next_word in apparitions_mod_k and current_position_mod_k in apparitions_mod_k[next_word]):
                necklace += letter
                break
        if next_word not in apparitions_mod_k:
            apparitions_mod_k[next_word] = []
        apparitions_mod_k[next_word].append(current_position_mod_k)
        current_position += 1
    return necklace
```

BIBLIOGRAPHY

[1] V. Becher and P. A. Herber. On extending de bruijn sequences. *Information Processing Letters*, 111:930–932, 2011.

[2] Verónica Becher and Olivier Carton. Normal numbers and nested perfect necklaces. *Journal of Complexity*, 54:101–403, 2019.

[3] Nicolaas G. de Bruijn. A combinatorial problem. Nederl. Akad. Wetensch., Proc.:758–764 = Indagationes Math. 8, 461–467 (1946), 1946.

[4] L. R. Ford. A cyclic arrangement of m-tuples. *Journal of Combinatorial Theory*, (Report P-1071), 1957.

[5] Harold Fredricksen. A survey of full length nonlinear shift register cycle algorithms. pages 195–221, 1982.

[6] Frank Harary. *Graph theory*. Addison-Wesley, 1969.

[7] Nicolás Álvarez, Verónica Becher, Pablo Ferrari, and Sergio Yuhjtman. Perfect necklaces. *Advances in Applied Mathematics*, 80:48–61, 2016.